LAPORAN PRAKTIKUM 4
PEMROGRAMAN LANJUT
ABSTRACT DATA TYPE

Oleh

Nama	:	Rizqillah
NIM	:	1957301020
Kelas	:	TI 2C
Dosen Pembimbing	:	Musta'inul Abdi, SST., M.Kom.

PROGRAM STUDI TEKNIK INFORMATIKA

JURUSAN TEKNOLOGI INFORMASI DAN KOMPUTER

TAHUN 2021

LEMBAR PENGESAHAN

No. Praktikum	:	04/PPL/2C/TI/2021
Judul	:	Abstract Data Type
Nama	:	Rizqillah
NIM / Kelas	:	1957301020 / TI 2C
Jurusan	:	Teknologi Informasi Dan Komputer
Prodi	:	Teknik Informatika
Tanggal praktikum	:	25 Maret 2021
Tanggal penyerahan	:	1 April 2021
Nilai	:	

Buketrata, 1 April 2021

Dosen Pembimbing,

Musta'inul Abdi, SST., M.Kom.
NIP. 19911030 20190310 1 5

DAFTAR ISI

BAB I
PENDAHULUAN

1.1 Tujuan

Modul ini mengenalkan suatu teknik pemrograman yang lebih tinggi. Dalam bagian ini Anda akan mempelajari Stack, Queue, dan Linked List.

Setelah menyelesaikan pelajaran ini, diharapkan Anda dapat:

1. Memahami dan menggunakan Stack, Queue dan Linked List
2. Membuat dan mengelola Stack, Queue dan Linked List dengan baik
3. Memahami cara menggunakan Stack, Queue dan Linked List

1.2 Dasar Teori

Abstract Data Type (ADT) adalah kumpulan dari elemen-elemen data yang disajikan dengan satu set operasi yang digambarkan pada elemen-elemen data tersebut. Stacks, queues dan binary trees adalah tiga contoh dari ADT. Dalam bab ini, Anda akan mempelajari tentang stacks dan queues.

Tipe data abstrak didefinisikan sebagai model matematika dari objek data yang membentuk sebuah tipe data,serta fungsi yang beroperasi pada objek-objek ini (Heilemen, 1996). Beberapa contoh dari tipe data abstrak, diantaranya adalah *Stack, Queue* dan *List*.

1.2.1 Stack

Stack adalah satu set atau urutan elemen data dimana manipulasi data dari elemen-elemen hanya diperbolehkan pada tumpukan teratas dari stack. Hal ini merupakan perintah pengumpulan data secara linier yang disebut “last in, first out” (LIFO). Stacks berguna untuk bermacam-macam aplikasi seperti pattern recognition dan pengkonversian antar notasi infix, postfix dan prefix .

Dua operasi yang dihubungkan dengan stacks adalah operasi push dan pop. Push berarti memasukkan data kedalam stacks yang paling atas dimana pop sebagai penunjuk/pointer untuk memindahkan elemen ke atas stacks. Untuk memahami bagaimana cara kerja stacks, pikirkan bagaimana Anda dapat menambah atau memindakan sebuah data dari tumpukan data. Pikiran Anda akan memberitahu Anda untuk menambah atau memindahkan data hanya pada stack yang paling atas karena jika menggunakan cara lain, dapat menyebabkan tumpukan stack akan terjatuh.

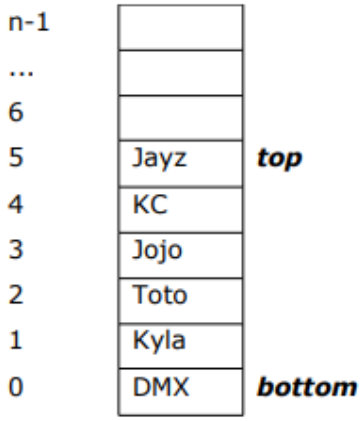
Kelebihan :

- membantu mengelola data dengan metode LIFO.
- secara otomatis membersihkan objek.
- tidak mudah rusak.
- ukuran variabel tidak dapat diubah.
- mengontrol memori secara mandiri.

Kekurangan :

- memori *stack* sangat terbatas.
- ada kemungkinan *stack* akan meluap atau *overflow* jika terlalu banyak objek.
- tidak memungkinkan akses acak, karena harus mengeluarkan tumpukan paling atas terlebih dahulu untuk mengakses tumpukan paling bawah.

Dibawah ini merupakan ilustrasi bagaimana tampilan dari stacks :



Tabel 1.2.2: Ilustrasi Stack

Stack akan berarti penuh jika jangkauan cell teratas disimbolkan dengan n-1. Jika nilai teratas / top sama dengan -1, stack berarti kosong.

Operasi Pada Stack :

- Pop, digunakan untuk mengeluarkan data teratas dari Stack.
- Push, digunakan untuk memasukkan data ke dalam Stack.
- isFull, digunakan untuk melihat apakah data penuh.
- siEmpty, digunakan untuk melihat apakah data kosong.
- Peek, digunakan untuk melihat data yang berada di posisi paling atas.
- Count, digunakan untuk mengetahui jumlah isi data pada Stack.
- Clear, digunakan untuk menghapus seluruh data yang ada pada Stack.
- Search, digunakan untuk mencari data Stack.
- Size, digunakan untuk melihat ukuran Stack.
- Show, digunakan untuk menampilkan seluruh isi Stack.

1.2.2 Queue

Queue adalah sekumpulan data yang mana penambahan elemen hanya bisa dilakukan pada suatu ujung yang disebut sisi belakang (rear), dan penghapusan (pengambilan elemen) dilakukan lewat bagian depan(front). Queues adalah contoh lain dari ADT. Hal ini merupakan perintah pengumpulan data yang disebut "first-in, first-out". Aplikasi ini meliputi jadwal pekerjaan dalam operating system, topological sorting dan graph traversal. Enqueue dan dequeue merupakan operasi yang dihubungkan dengan queues. Enqueu menunjuk pada memasukkan data pada akhir queue dimana dequeue berarti memindahkan elemen dari queue tersebut. Untuk mengingat bagaimana queue bekerja, ingatlah arti khusus dari queue yaitu baris.

Operasi Pada Queue :

- Enqueue yang berfungsi menambah sebuah elemen ke dalam antrian.
- Dequeue yang berfungsi menghapus atau mengeluarkan elemen dari antrian.
- isFull yang berfungsi melihat apakah data queue full atau tidak.
- isEmpty yang berfungsi melihat apakah data queue masih kosong.
- Size yang berfungsi melihat panjang data queue.
- Show yang berfungsi menampilkan seluruh data queue.
- Clear yang berfungsi menghapus seluruh data queue.

Dalam ilmu komputer, antrian atau queue banyak digunakan terutama dalam sistem operasi yang memerlukan manajemen sumber daya dan penjadwalan. Contohnya time-sharing computer-system yang bisa dipakai oleh sejumlah orang secara serempak. Di dalam sebuah antrian (queue), terdapat sebuah operasi bernama add_priority. Dalam hal ini, antrian tidak lagi menerapkan konsep antrian yang murni, namun berubah menjadi antrian dengan prioritas. Dimana terdapat prioritas tertentu pada elemen, dan elemen yang baru ditambah tidak mesti berada di akhir.

Berikut ini merupakan ilustrasi dari bagaimana tampilan dari queue :

0	1	2	3	4	5	6	7	8	9	...	n-1
		Eve	Jayz	KC	Jojo	Toto	Kyla	DMX			
		front						***end***	☐	Insert	
									☐	Delete	

Tabel 1.2.3: Ilustrasi Queue

Queue akan kosong jika nilai end kurang dari front. Sementara itu, akan penuh jika end sama dengan n-1.

1.3 Sequential dan Linked Representation

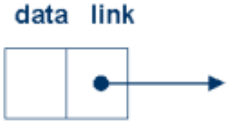
ADT biasanya dapat diwakilkan menggunakan sequential dan linked representation. Hal ini memudahkan untuk membuat sequential representation dengan menggunakan array. Bagaimanapun juga, masalah dengan menggunakan array adalah pembatasan size, yang membuatnya tidak fleksibel. Dengan menggunakan array, sering terjadi kekurangan atau kelebihan space memory. Mempertimbangkan hal tersebut, Anda harus membuat sebuah array dan mendeklarasikannya agar mampu menyimpan 50 elemen. Jika user hanya memasukkan 5 elemen, maka 45 space pada memory akan sia-sia. Disisi lain, jika user ingin memasukkan 51 elemen, space yang telah disediakan didalam array tidak akan cukup. Dibandingkan dengan sequential representation, linked representation lebih sedikit rumit tetapi lebih fleksibel. Linked representation menyesuaikan memory yang dibutuhkan oleh user.

1.3.1 Linked List

Linked List adalah suatu struktur data linier. Berbeda dengan array yang juga merupakan struktur data linier dan tipe data komposit, linked list dibentuk secara dinamik. Pada saat awal program dijalankan elemen linked list belum ada data. Elemen linked list (disebut node) dibentuk sambil jalan sesuai instruksi. Apabila setiap elemen array dapat diakses secara langsung dengan menggunakan indeks, sebuah node linked list diakses dengan menggunakan pointer yang mengacu (menunjuk) ke node tersebut. Awal atau kepala linked list harus diacu sebuah pointer yang biasa diberi nama head. Pointer current (disingkat curr) digunakan untuk memindahkan pengacuan kepada node tertentu.

Linked list merupakan struktur dinamis yang berlawanan dengan array, yang merupakan struktur statis. Hal ini berarti linked list dapat tumbuh dan berkurang dalam size yang bergantung pada kebutuhan user. Linked list digambarkan sebagai kumpulan dari nodes, Yang masing-masing berisi data dan link atau pointer ke node berikutnya didalam list.

Gambar dibawah ini menunjukkan tampilan dari node.



Berikut ini merupakan contoh dari non-empty linked list dengan 3 node.

BAB II
PEMBAHASAN

1.4 Stack

Program Stack dibawah ini adalah program yang menggunakan penyimpanan data melalui array bertipe integer.

1.4.1 Integer Stack

Program :

```
class SeqStack {
    int top = -1;
    /* initially, the stack is empty */
    int memSpace[];
    /* storage for integers */
    int limit;
    /* size of memSpace */

    SeqStack() {
        memSpace = new int[10];
        limit = 10;
    }

    SeqStack(int size) {
        memSpace = new int[size];
        limit = size;
    }

    public boolean isEmpty() {
        if (top == -1) {
            return true;
        } else {
            return false;
        }
    }

    public boolean isFull() {
        if (top == memSpace.length - 1) {
            return true;
        } else {
            return false;
        }
    }

    public int length() {
        return top + 1;
    }
```

```
    boolean push(int value) {
        top++;
        /* check if the stack is full */
        if (top < limit) {
            memSpace[top] = value;
        } else {
            top--;
            return false;
        }
        return true;
    }

    int pop() {
        int temp = -1;
        /* check if the stack is empty */
        if (top >= 0) {
            temp = memSpace[top];
            top--;
        } else {
            return -1;
        }
        return temp;
    }

    public static void main(String args[]) {
        SeqStack myStack = new SeqStack(3);
        myStack.push(1);
        myStack.push(2);
        myStack.push(3);
        myStack.push(4);
        System.out.println("Panjang Stack : " + myStack.length());
        System.out.print("{" + myStack.pop() + "}");
        System.out.print("{" + myStack.pop() + "}");
        System.out.print("{" + myStack.pop() + "}");
        System.out.print("{" + myStack.pop() + "}");
        System.out.println("\nPanjang Stack : " + myStack.length());
    }
}
```

Hasil :

```
Panjang Stack : 3
{3}{2}{1}{-1}
Panjang Stack : 0
BUILD SUCCESSFUL (total time: 0 seconds)
```

1.4.2 Linked List Stack

Program Stack dibawah ini adalah program yang menggunakan penyimpanan data melalui Linked List.

Program :

```
public class NodeStack {
    private Node top;
    private int size;

    private class Node {
        private int data;
        private Node next;
        public Node(int data) {
            this.data = data;
            this.next = null;
        }
    }

    public void push(int data) {
        Node tempNode = new Node(data);
        tempNode.next = top;
        top = tempNode;
        size++;
    }

    public int pop() {
        if (isEmpty()) {
            return -1;
        }
        int result = top.data;
        top = top.next;
        size--;
        return result;
    }

    public int peek() {
        if (isEmpty()) {
            return -1;
        }
        return top.data;
    }
    public int size() {
        return size;
    }
```

```
    public boolean isEmpty() {
        return size == 0;
    }

    public void show() {
        Node current = top;

        while (current != null) {
            System.out.print(current.data + " ");
            current = current.next;
        }
    }
}

class main {
    public static void main(String[] args) {
        NodeStack stack = new NodeStack();
        System.out.println("Panjang Stack : " + stack.size());
        System.out.println("Stack Kosong? : " + stack.isEmpty());
        stack.push(10);
        stack.push(20);
        stack.push(30);
        stack.push(40);
        stack.push(50);

        System.out.println("Panjang Stack : " + stack.size());
        System.out.println("Stack Kosong? : " + stack.isEmpty());
        System.out.println("Peek nilai Stack : " + stack.peek());
        System.out.println("Keluarkan nilai Stack : " +
stack.pop());
        System.out.println("Panjang Stack : " + stack.size());
    }
}
```

Hasil :

```
Panjang Stack : 0
Stack Kosong? : true
Panjang Stack : 5
Stack Kosong? : false
Peek nilai Stack : 50
Keluarkan nilai Stack : 50
Panjang Stack : 4
BUILD SUCCESSFUL (total time: 0 seconds)
```

Analisis :

`public class NodeStack {`

➔ **Membuat class bernama NodeStack**

```
private Node top;
private int size;
```
➔ **Deklarasi variabel global top bertipe data Node dan size bertipe data int**

```
private class Node {
```
➔ **Membuat class private bernama Node**

```
  private int data;
  private Node next;
```
➔ **Deklarasi variabel data bertipe int dan variabel next bertipe data class Node**

```
  public Node(int data) {
```
➔ **Deklarasi method construct yang menerima parameter variabel data bertipe int**

```
      this.data = data;
      this.next = null;
```
➔ **Mengisi nilai variabel global data dengan nilai dari parameter data. Dan membuat nilai variabel global next bernilai null**

```
  }
}

public void push(int data) {
```
➔ **Membuat method push dengan tipe void dan menerima parameter data dengan tipe int**

```
  Node tempNode = new Node(data);
```
➔ **Membuat objek dari class Node dengan nama tempNode dan mengisi nilai parameter construct dengan nilai variabel data**

```
  tempNode.next = top;
```
➔ **Mengisi nilai variabel pada class Node melalui objek yang telah dibuat ke variabel next, mengisi dengan nilai dari top**

```
  top = tempNode;
```
➔ **Mengisi nilai top menjadi nilai tempNode**

```
  size++;
```
➔ **Melakukan increment terhadap nilai variabel size**

```
}

public int pop() {
```
➔ **Membuat method pop dengan tipe integer**

```
  if (isEmpty()) {
```
➔ **Melakukan pengecekan, apakah nilai di method isEmpty = true**

```
        return -1;
```
➔ **Jika iya, maka mengembalikan angka -1 ke si pemanggil method pop**
```
    }
    int result = top.data;
```
➔ **Membuat variabel result yang dimasukan nilai dari data melalui variabel top**
```
    top = top.next;
```
➔ **Membuat pointer top akan bergerak ke nilai selanjutnya**
```
    size--;
    return result;
```
➔ **Melakukan decrement terhadap variabel size. Dan mengembalikan nilai result ke yang memanggil method pop**
```
}

public int peek() {
```
➔ **Membuat method peek dengan tipe int**
```
    if (isEmpty()) {
        return -1;
```
➔ **Mengecek apakah hasil kembalian method isEmpty = true, jika iya akan di return nilai -1 ke yang memanggil method peek**
```
    }
    return top.data;
```
➔ **Melakukan pengembalian nilai data melalui variabel top ke yang memanggil method peek**
```
}

public int size() {
    return size;
```
➔ **Membuat method size yang akan mengembalikan nilai dari variabel size**
```
}

public boolean isEmpty() {
    return size == 0;
```
➔ **Membuat method isEmpty dengan tipe boolean yang akan mengembalikan nilai true atau false ketika pengecekan nilai size == 0. Dan jika nilai size tidak sama dengan 0, maka mengembalikan false**
```
}
```

```
    public void show() {
        Node current = top;
```
➔ **Membuat method show. Kemudian membuat variabel current dari Node dengan nilai dari top**

```
        while (current != null) {
```
➔ **Mengecek selama nilai dari current tidak sama dengan null, maka akan terus-terusan melakukan perulangan**

```
            System.out.print(current.data + " ");
            current = current.next;
```
➔ **Mencetak nilai dari data melalui current. Kemudian mengisi nilai current dengan nilai lanjutan**

```
        }
    }
}

class main {
    public static void main(String[] args) {
        NodeStack stack = new NodeStack();
```
➔ **Membuat objek dari class NodeStack dengan nama stack**

```
        System.out.println("Panjang Stack : " + stack.size());
```
➔ **Mencetak nilai dari method size melalui objek stack**

```
        System.out.println("Stack Kosong? : " + stack.isEmpty());
```
➔ **Mencetak nilai dari method isEmpty**

```
        stack.push(10);
        stack.push(20);
        stack.push(30);
        stack.push(40);
        stack.push(50);
```
➔ **Memanggil method push dan mengisi parameter method push**

```
        System.out.println("Panjang Stack : " + stack.size());
        System.out.println("Stack Kosong? : " + stack.isEmpty());
        System.out.println("Peek nilai Stack : " + stack.peek());
```
➔ **Mengintip nilai yang ada dibarisan terakhir dari stack**

```
        System.out.println("Keluarkan nilai Stack : " +
stack.pop());
```
➔ **Mengambil dan mengeluarkan nilai stack yang ada di bagian terakhir**

```
        System.out.println("Panjang Stack : " + stack.size());
```
➔ **Mencetak ukuran dari stack yang sekarang**
```
    }
}
```

Pembahasan :

Dari kedua program diatas, program yang menggunakan metode stack dengan penyimpanan array jelas lebih tidak praktis, dikarenakan data yang bisa disimpan terbatas, dan solusi dalam penyimpanan data yang bebas adalah menggunakan metode penyimpanan melalui linked list. Adapun array adalah suatu struktur data statis, dan linked list adalah struktur data yang dinamis.

1.5 Queue

Program Queue dibawah ini adalah program yang menggunakan penyimpanan data melalui Linked List.

Program :

```
public class NodeQueue<Tipe> {
    private Node<Tipe> front;
    private Node<Tipe> rear;
    private int length;

    private static class Node<Tipe> {
        private final Tipe data;
        private Node<Tipe> next;
        public Node(Tipe data) {
            this.data = data;
        }
    }

    /**
     * Method to EnQueue or insert an Item in Queue(rear)
     * @param data Item to be inserted in Queue
     */
    public void enQueue(Tipe item) {
        if (front == null) {
            rear = new Node<Tipe>(item);
            front = rear;
        } else {
            rear.next = new Node<Tipe>(item);
            rear = rear.next;
        }
        length++;
    }
```

```
    /**
     * Method to DeQueue or Remove an Item From Queue
     * @return return DeQueued or Removed Item from queue(front)
     */
    public Tipe deQueue() {
        if (front != null) {
            Tipe item = front.data;
            front = front.next;
            length--;
            return item;
        }
        return null;
    }

    public int size() {
        return length;
    }

    public void showQueue() {
        Node<Tipe> currentNode = front;
        while (currentNode != null) {
            System.out.print(currentNode.data + " ");
            currentNode = currentNode.next;
        }
    }
}

class mainqueue {
    public static void main(String[] args) {
        NodeQueue<Integer> queue = new NodeQueue<>();
        queue.enQueue(10);
        queue.enQueue(20);
        queue.enQueue(30);
        queue.enQueue(40);
        queue.enQueue(50);
        System.out.println("Panjang Queue : " + queue.size());
        queue.showQueue();
        System.out.println();
        System.out.println("Nilai Dequeue :" + queue.deQueue());
        System.out.println("Panjang Queue :" + queue.size());
        System.out.println("--------------------");
        queue.showQueue();
        System.out.println();
    }
}
```

Hasil :

```
Panjang Queue : 5
10 20 30 40 50
Nilai Dequeue :10
Panjang Queue :4
-------------------
20 30 40 50
BUILD SUCCESSFUL (total time: 0 seconds)
```

Analisis :

```
public class NodeQueue<Tipe> {
    private Node<Tipe> front;
    private Node<Tipe> rear;
    private int length;
```

➔ **Membuat class NodeQueue yang memiliki penerimaan nama tipe data**

➔ **Membuat variabel front dan rear yang memiliki tipe data dari class Node yang mempunyai pengambilan nama tipe data**

➔ **Membuat variabel length untuk menghitung panjang dari queue nantinya**

```
    private static class Node<Tipe> {
        private final Tipe data;
        private Node<Tipe> next;
```

➔ **Membuat class Node yang mempunyai pengambilan tipe data**

➔ **Membuat variabel data yang memiliki tipe data dari yang diinput user**

➔ **Membuat variabel next yang bertipe Node dan memiliki inputan tipe data**

```
        public Node(Tipe data) {
            this.data = data;
```

➔ **Membuat construct yang memiliki parameter data dengan tipe yang diambil dari inputan**

➔ **Mengisi nilai data yang di global menjadi nilai data yang dari local**

```
        }
    }
    public void enQueue(Tipe item) {
```

➔ **Membuat method enQueue yang memiliki parameter item dengan tipe dari inputan**

```
        if (front == null) {
```

➔ **Mengecek apakah hasil dari nilai front == null, jika iya akan menghasilkan true**

```
        rear = new Node<Tipe>(item);
        front = rear;
```
➔ **Mengisi nilai variabel rear dengan nilai dari class Node yang diisi nilai item**
➔ **Membuat nilai front menjadi nilai rear**
```
    } else {
        rear.next = new Node<Tipe>(item);
        rear = rear.next;
```
➔ **Jika hasil pengecekan bernilai false maka akan membuat nilai next melalui rear menjadi bernilai dari class Node yang diisi nilai item**
➔ **Dan nilai rear menjadi nilai dari next melalui variabel rear**
```
    }
    length++;
```
➔ **Melakukan increment variabel length**
```
}

public Tipe deQueue() {
```
➔ **Membuat method deQueue dengan tipe data dari deklarasi objek nantinya**
```

    if (front != null) {
```
➔ **Mengecek apakah nilai pada variabel front tidak null**
```

        Tipe item = front.data;
```
➔ **Mengisi nilai pada variabel item dengan nilai dari data melalui variabel front**
```
        front = front.next;
        length--;
        return item;
```
➔ **Mengisi nilai pada variabel front dengan nilai dari next yang berarti nilai selanjutnya**
➔ **Melakukan decrement pada variabel length**
➔ **Mengembalikan nilai dari variabel item ke yang memanggil method deQueue**
```
    }
    return null;
```
➔ **Melakukan return null ke si pemanggil method deQueue**
```
}

public int size() {
    return length;
}
```
➔ **Membuat method size dengan tipe data int**
➔ **Mengembalikan nilai variabel length**

```
    public void showQueue() {
```
➔ **Membuat method showQueue dengan tipe void**

```
        Node<Tipe> currentNode = front;
```
➔ **Membuat variabel dari class Node dengan nama currentNode dan diisi nilai dari front**

```
        while (currentNode != null) {
```
➔ **Melakukan perulangan selama nilai currentNode tidak sama dengan null**

```
            System.out.print(currentNode.data + " ");
            currentNode = currentNode.next;
```
➔ **Mencetak nilai variabel data melalu objek currentNode**

➔ **Mengisi nilai currentNode menjadi nilai next atau nilai selanjutnya**

```
        }
    }
}

class mainqueue {
    public static void main(String[] args) {
        NodeQueue<Integer> queue = new NodeQueue<>();
```
➔ **Membuat objek dari class NodeQueue dengan nama queue beserta mendeklarasi tipe data Integer**

```
        queue.enQueue(10);
        queue.enQueue(20);
        queue.enQueue(30);
        queue.enQueue(40);
        queue.enQueue(50);
```
➔ **Melakukan operasi pemasukan data ke Queue**

```
        System.out.println("Panjang Queue : " + queue.size());
        queue.showQueue();
        System.out.println();
```
➔ **Melihat ukuran queue, dan menampilkan seluruh nilai queue**

```
        System.out.println("Nilai Dequeue :" + queue.deQueue());
        System.out.println("Panjang Queue :" + queue.size());
        System.out.println("--------------------");
        queue.showQueue();
        System.out.println();
```
➔ **Mencetak hasil pengambilan nilai queue, dan ukuran queue setelahnya, kemudian menampilkan seluruh isi queue**

```
    }
}
```

1.6 Linked List

Program :

```
class Node {
    int data;
    /* integer data contained in the node */
    Node nextNode;
    /* the next node in the list */
}

class TestNode {
    public static void main(String args[]) {
        Node head = new Node();
        /* initialize 1st node in the list */
        head.data = 5;
        head.nextNode = new Node();
        head.nextNode.data = 10;
        /* null marks the end of the list */
        head.nextNode.nextNode = null;
        /* print elements of the list */
        Node currNode = head;
        while (currNode != null) {
            System.out.println(currNode.data);
            currNode = currNode.nextNode;
        }
    }
}
```

Hasil :

```
5
10
BUILD SUCCESSFUL (total time: 0 seconds)
```

Analisis :

```
class Node {
```

➔ **Membuat class Node**

```
    int data;
    /* integer data contained in the node */
    Node nextNode;
    /* the next node in the list */
```

➔ **Membuat variabel data dengan tipe int. Dan variabel nextNode dengan tipe data non-primitif yaitu dari class Node, yang nantinya berguna sebagai untuk merubah penunjuk pointer**

```
}
```

```
class TestNode {
    public static void main(String args[]) {
        Node head = new Node();
        /* initialize 1st node in the list */
```

➔ **Membuat objek dari class Node dengan nama head**

```
        head.data = 5;
```

➔ **Menyimpan nilai 5 pada variabel data dalam class Node melalui objek yang telah dibuat**

```
        head.nextNode = new Node();
```

➔ **Membuat objek baru melalui objek baru dengan cara memanggil nextNode melalui objek dan mengisi data dengan nilai objek baru**

```
        head.nextNode.data = 10;
```

➔ **Mengisi nilai data melalui nextNode dan head. Dengan nilai 10**

```
        /* null marks the end of the list */
        head.nextNode.nextNode = null;
```

➔ **Membuat nilai nextNode selanjutnya sebagai null agar ketika melakukan perulangan mencetak nilai data, maka akan berhenti ketika menemukan null**

```
        /* print elements of the list */
        Node currNode = head;
```

➔ **Membuat variabel dari class Node dengan nama currNode dan diisi dengan nilai objek head. Dikarenakan variabel yang dibuat memiliki tipe data non-primitif, oleh karenanya hal ini bisa terjadi**

```
        while (currNode != null) {
```

➔ **Melakukan perulangan selama nilai dari currNode tidak sama dengan null**

```
            System.out.println(currNode.data);
            currNode = currNode.nextNode;
```

➔ **Mencetak nilai data melalui currNode. Kemudian mengisi nilai currNode dengan nilai nextNode. Proses ini akan membuat pointer berada pada nilai selanjutnya**

```
        }
    }
}
```

1.7 LATIHAN

1.7.1 Faktor Persekutuan Terbesar

Faktor persekutuan terbesar (FPB) dari dua angka adalah angka yang terbesar selalu dibagi oleh angka yang satunya, kemudian modulus atau sisa pembagian membagi angka kedua dan seterusnya hingga sisa pembagian dari kedua angka tersebut sama dengan nol. Menggunakan Euclid's method, buatlah dua kode untuk penghitungan dua angka. Gunakan iterasi untuk kode program yang pertama dan rekursif untuk kode program berikutnya.

Catatan pada algoritma Euclid :

1. Sebagai masukkan integers x dan y.
2. Ulangi step dibawah ini while y != 0
   a. y = x % y;
   b. x = Nilai lama y;
3. Return x.

Contoh, x = 14 dan y = 6.
y = x % y = 14 % 6 = 2
x = 6
y = x % y = 6 % 2 = 0
x = 2 (FPB)

Program Iterasi :

```
import java.util.Scanner;
public class FPB {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        int x, y, fpb;
        System.out.print("Input bilangan x : ");
        x = input.nextInt();
        System.out.print("Input bilangan y : ");
        y = input.nextInt();

        fpb = x % y;
        while (fpb != 0) {
            x = y;
            y = fpb;
            fpb = x % y;
        }
        System.out.println("Nilai FPB sebagai berikut : " + y);
    }
}
```

Hasil :

```
Input bilangan x : 14
Input bilangan y : 6
Nilai FPB sebagai berikut : 2
BUILD SUCCESSFUL (total time: 4 seconds)
```

Program Recursif :

```
import java.util.Scanner;
public class FPBRecursif {
    static int fpb(int x, int y) {
        int fpb = x % y;
        if (fpb != 0) {
            x = y;
            y = fpb;
            fpb = fpb(x, y);
        }
        return y;
    }

    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        int x, y;
        System.out.print("Input bilangan x : ");
        x = input.nextInt();
        System.out.print("Input bilangan y : ");
        y = input.nextInt();

        System.out.println("Nilai FPB : " + fpb(x, y));
    }
}
```

Hasil :

```
Input bilangan x : 14
Input bilangan y : 6
Nilai FPB : 2
BUILD SUCCESSFUL (total time: 5 seconds)
```

Analisis :

Iterasi :

```
import java.util.Scanner;
public class FPB {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        int x, y, fpb;
```

➔ **Membuat variabel Scanner dengan nama input**

➔ **Membuat variabel x, y dan fpb dengan tipe data int**

```
        System.out.print("Input bilangan x : ");
        x = input.nextInt();
        System.out.print("Input bilangan y : ");
        y = input.nextInt();
```
➔ **Meminta inputan dari user yang akan disimpan di variabel x dan y**

```
        fpb = x % y;
```
➔ **Melakukan modulus antara x % y, hasilnya disimpan dalam variabel fpb**

```
        while (fpb != 0) {
```
➔ **Melakukan perulangan selama nilai fpb tidak sama dengan 0**

```
            x = y;
            y = fpb;
            fpb = x % y;
```
➔ **Membuat nilai x menjadi nilai y**
➔ **Membuat nilai y menjadi nilai fpb**
➔ **membuat nilai fpb menjadi nilai dari hasil x % y(x mod y)**

```
        }
        System.out.println("Nilai FPB sebagai berikut : " + y);
```
➔ **mencetak kalimat beserta nilai y**

```
    }
}
```

Recursif :

```
import java.util.Scanner;
public class FPBRecursif {
    static int fpb(int x, int y) {
```
➔ **Membuat method fpb dengan tipe data int dan menerima parameter variabel x dan y yang bertipe int**

```
        int fpb = x % y;
```
➔ **Membuat variabel fpb yang mengisi nilai dari hasil x mod y**

```
        if (fpb != 0) {
```
➔ **Melakukan pengecekan apakah nilai fpb tidak sama dengan 0**

```
            x = y;
            y = fpb;
```
➔ **Membuat nilai x menjadi nilai y**
➔ **Dan membuat nilai y menjadi nilai fpb**

```
            fpb = fpb(x, y);
```
➔ **Membuat nilai fpb menjadi nilai dari hasil pemanggilan fungsi fpb yang diisi nilai x dan y pada pada parameter fungsi**

```
        }
        return y;
```
➔ **Mengembalikan hasil nilai y ke si pemanggil fungsi fpb**

```
    }
```

```
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        int x, y;
```

➔ **Membuat variabel dari Scanner dengan nama input**

➔ **Membuat variabel x dan y yang bertipe int**

```
        System.out.print("Input bilangan x : ");
        x = input.nextInt();
        System.out.print("Input bilangan y : ");
        y = input.nextInt();
```

➔ **Meminta inputan dari user yang akan disimpan dalam variabel x dan y**

```
        System.out.println("Nilai FPB : " + fpb(x, y));
```

➔ **Mencetak nilai dari hasil pemanggilan fungsi fpb yang diisi dengan nilai parameter x dan y**

```
    }
}
```

1.7.2 Sequential Representation dari Integer Queue

Dengan menggunakan array, implementasikan sebuah integer queue seperti contoh pada sequential stack.

Program :

```
public class Queue {
    int data[];
    int head;
    int tail = -1;

    public Queue(int size) {
        data = new int[size];
    }

    public boolean isEmpty() {
        if (tail == -1) {
            return true;
        } else {
            return false;
        }
    }

    public boolean isFull() {
        if (tail == data.length - 1) {
            return true;
        } else {
            return false;
        }
    }
```

```
    public void Enqueue(int dataBaru) {
        if (isEmpty()) {
            tail = head;
            data[tail] = dataBaru;
        } else if (!isFull()) {
            tail++;
            data[tail] = dataBaru;
        } else if (isFull()) {
            System.out.println("antrian sudah penuh");
        }
    }

    public int Dequeue() {
        int temp = data[head];
        for (int i = head; i <= tail - 1; i++) {
            data[i] = data[i + 1];
        }
        tail--;
        return temp;
    }

    public void show() {
        if (!isEmpty()) {
            int index = head;
            while (index <= tail) {
                System.out.print("{" + data[index] + "} ");
                index++;
            }
            System.out.println();
        } else {
            System.out.println("Kosong");
        }
    }

    public static void main(String[] args) {
        Queue queue = new Queue(3);
        queue.Enqueue(1);
        queue.Enqueue(2);
        queue.Enqueue(3);
        queue.show();
        queue.Dequeue();
        queue.show();
        queue.Dequeue();
        queue.show();
        queue.Dequeue();
        queue.show();
    }
}
```

Hasil :

```
{1} {2} {3}
{2} {3}
{3}
Kosong
BUILD SUCCESSFUL (total time: 0 seconds)
```

Analisis :

```
public class Queue {
```

➔ **Membuat class dengan nama Queue**

```
 int data[];
 int head;
 int tail = -1;
```

➔ **Membuat variabel array dengan nama data, head dan tail dengan nilai -1 dengan tipe data int**

```
 public Queue(int size) {
     data = new int[size];
 }
```

➔ **Membuat construct dengan menerima parameter size dengan tipe data int**

➔ **Mengisi nilai panjang array pada variabel data dengan nilai dari size**

```
 public boolean isEmpty() {
```

➔ **Membuat method isEmpty bertipe boolean**

```
     if (tail == -1) {
         return true;
```

➔ **Mengecek apakah nilai tail sama dengan -1, jika iya maka akan mengembalikan true. Yang berarti nilai masih kosong**

```
     } else {
         return false;
```

➔ **Jika tidak, maka akan mengembalikan false ke yang memanggil method isEmpty**

```
     }
 }

 public boolean isFull() {
```

➔ **Membuat method isFull dengan tipe data boolean**

```
     if (tail == data.length - 1) {
         return true;
```

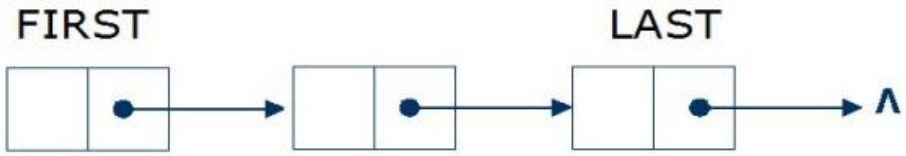
➔ **Mengecek apakah nilai tail sama dengan dengan nilai panjang data – 1, jika iya maka akan di return true**

```
    } else {
        return false;
```
➔ **Jika tidak, akan di return false ke yang memanggil method isFull**
```
    }
}

public void Enqueue(int dataBaru) {
```
➔ **Membuat method enqueue dengan tipe void yang menerima nilai parameter dataBaru yang bertipe int**
```
    if (isEmpty()) {
        tail = head;
        data[tail] = dataBaru;
```
➔ **Mengecek apakah nilai kembalikan pemanggilan method isEmpty true, jika iya maka nilai tail akan diisi dengan nilai head, nilai head secara default adalah 0.**
➔ **Kemudian mengisi nilai dataBaru pada array data yang berada di index tail(0)**
```
    } else if (!isFull()) {
        tail++;
        data[tail] = dataBaru;
```
➔ **Melakukan pengecekan apakah hasil kembalian method isFull bukan true, jika iya maka nilai tail dilakukan increment**
➔ **Dan nilai dataBaru diisi pada array data di index nilai tail sekarang**
```
    } else if (isFull()) {
        System.out.println("antrian sudah penuh");
```
➔ **Mengecek apakah hasil kembalian isFull true, yang berarti nilai sudah penuh. Dan mencetak bahwa antrian sudah penuh**
```
    }
}

public int Dequeue() {
```
➔ **Membuat method Dequeue bertipe int**
```
    int temp = data[head];
```
➔ **Membuat variabel temp yang diisi nilai array data yang berada di index head yang berarti index 0**
```
    for (int i = head; i <= tail - 1; i++) {
        data[i] = data[i + 1];
    }
```
➔ **Melakukan perulangan selama nilai i = head(i=0), dan i <= tail-1, jika iya maka**
➔ **Nilai array data pada index i+1 akan diisi pada array di index i**

```
        tail--;
        return temp;
```
➔ **Melakukan decrement terhadap variabel tail**
➔ **Mengembalikan nilai temp ke yang memanggil method Dequeue**
```
    }

    public void show() {
```
➔ **Membuat method show bertipe void**
```

        if (!isEmpty()) {
            int index = head;
```
➔ **Mengecek apakah hasil kembalian method isEmpty bukan true, maka membuat variabel index yang diisi nilai dari head atau 0**
```

            while (index <= tail) {
                System.out.print("{" + data[index] + "} ");
                index++;
```
➔ **Melakukan perulangan while selama nilai index lebih kecil atau sama dengan nilai tail**
➔ **Mencetak nilai array data di index dalam nilai index**
➔ **Melakukan increment terhadap variabel index**
```
            }
            System.out.println();
```
➔ **Mencetak sout kosong sebagai baris baru**
```
        } else {
            System.out.println("Kosong");
```
➔ **Mencetak kata bahwa queue masih kosong**
```
        }
    }

    public static void main(String[] args) {
        Queue queue = new Queue(3);
```
➔ **Membuat objek dari class Queue dengan nama queue dan diisi nilai parameter 3 pada method construct**
```

        queue.Enqueue(1);
        queue.Enqueue(2);
        queue.Enqueue(3);
```
➔ **Memanggil method Enqueue melalui objek dan mengisi nilai parameter**
```
        queue.show();
```
➔ **Memanggil method show melalui objek**
```
        queue.Dequeue();
```
➔ **Memanggil method dequeue melalui objek**
```
        queue.show();
        queue.Dequeue();
        queue.show();
        queue.Dequeue();
        queue.show();
    }
}
```

1.7.3 Linked Representation dari Integer Queue

Dengan menggunakan ide dari linked list, implementasikan sebuah integer queue dinamis seperti integer stack dinamis yang diperkenalkan seperti contoh berikut.



Program :

```
public class NodeIntQueue {
    private Node front;
    private Node rear;
    private int length;

    private static class Node {

        private final int data;
        private Node next;

        public Node(int data) {
            this.data = data;
            this.next = null;
        }
    }

    public void enQueue(int item) {
        if (front == null) {
            rear = new Node(item);
            front = rear;
        } else {
            rear.next = new Node(item);
            rear = rear.next;
        }
        length++;
    }

    public int deQueue() {
        if (front != null) {
            int item = front.data;
            front = front.next;
            length--;
            return item;
        }
        return -1;
    }

    public int size() {
        return length;
    }
```

```
    public void showQueue() {
        Node currentNode = front;
        while (currentNode != null) {
            System.out.print(currentNode.data + " ");
            currentNode = currentNode.next;
        }
    }
}

class mainqueueint {
    public static void main(String[] args) {
        NodeQueue queue = new NodeQueue();
        queue.enQueue(10);
        queue.enQueue(20);
        queue.enQueue(30);
        queue.enQueue(40);
        queue.enQueue(50);

        System.out.println("Panjang Queue : " + queue.size());
        queue.showQueue();
        System.out.println();
        System.out.println("Nilai Dequeue :" + queue.deQueue());
        System.out.println("Panjang Queue :" + queue.size());
        System.out.println("--------------------");
        queue.showQueue();
        System.out.println();
    }
}
```

Hasil :

```
Panjang Queue : 5
10 20 30 40 50
Nilai Dequeue :10
Panjang Queue :4
--------------------
20 30 40 50
BUILD SUCCESSFUL (total time: 0 seconds)
```

Analisis :

```
public class NodeIntQueue {
    private Node front;
    private Node rear;
    private int length;
```

- ➔ **Membuat class NodeIntQueue**
- ➔ **Membuat variabel front dan rear yang memiliki tipe data Node**
- ➔ **Membuat variabel length untuk menghitung panjang dari queue nantinya**

```
 private static class Node {
     private final int data;
     private Node next;
```
➔ **Membuat class Node**
➔ **Membuat variabel data yang memiliki tipe data int dan nilainya tetap**
➔ **Membuat variabel next yang bertipe Node**

```
     public Node(int data) {
         this.data = data;
         this.next = null;
```
➔ **Membuat construct yang memiliki parameter int data**
➔ **Mengisi nilai data yang di global menjadi nilai data yang dari local**
➔ **Mengisi nilai variabel next menjadi null**

```
     }
 }

 public void enQueue(int item) {
```
➔ **Membuat method enQueue yang memiliki parameter item dengan tipe data int**

```
     if (front == null) {
```
➔ **Mengecek apakah hasil dari nilai front == null, jika iya akan menghasilkan true**

```
         rear = new Node(item);
         front = rear;
```
➔ **Mengisi nilai variabel rear dengan nilai dari class Node yang diisi nilai item**
➔ **Membuat nilai front menjadi nilai rear**

```
     } else {
         rear.next = new Node(item);
         rear = rear.next;
```
➔ **Jika hasil pengecekan bernilai false maka akan membuat nilai next melalui rear menjadi bernilai dari class Node yang diisi nilai item**
➔ **Dan nilai rear menjadi nilai dari next melalui variabel rear**

```
     }
     length++;
```
➔ **Melakukan increment variabel length**

```
 }

 public int deQueue() {
```
➔ **Membuat method deQueue dengan tipe data int**

```
     if (front != null) {
```
➔ **Mengecek apakah nilai pada variabel front tidak null**

```
         int item = front.data;
```
➔ **Mengisi nilai pada variabel item dengan nilai dari data melalui variabel front**

```
            front = front.next;
            length--;
            return item;
```
➔ **Mengisi nilai pada variabel front dengan nilai dari next yang berarti nilai selanjutnya**

➔ **Melakukan decrement pada variabel length**

➔ **Mengembalikan nilai dari variabel item ke yang memanggil method deQueue**

```
        }
        return -1;
```
➔ **Melakukan return -1 ke si pemanggil method deQueue**

```
    }

    public int size() {
        return length;
    }
```
➔ **Membuat method size dengan tipe data int**

➔ **Mengembalikan nilai variabel length**

```
    public void showQueue() {
```
➔ **Membuat method showQueue dengan tipe void**

```
        Node currentNode = front;
```
➔ **Membuat variabel dari class Node dengan nama currentNode dan diisi nilai dari front**

```
        while (currentNode != null) {
```
➔ **Melakukan perulangan selama nilai currentNode tidak sama dengan null**

```
            System.out.print(currentNode.data + " ");
            currentNode = currentNode.next;
```
➔ **Mencetak nilai variabel data melalu objek currentNode**

➔ **Mengisi nilai currentNode menjadi nilai next atau nilai selanjutnya**

```
        }
    }
}

class mainqueueint {
    public static void main(String[] args) {
        NodeQueue queue = new NodeQueue();
```
➔ **Membuat objek dari class NodeQueue dengan nama queue beserta mendeklarasi tipe data Integer**

```
        queue.enQueue(10);
        queue.enQueue(20);
        queue.enQueue(30);
        queue.enQueue(40);
        queue.enQueue(50);
```
➔ **Melakukan operasi pemasukan data ke Queue dengan method enQueue**

```
            System.out.println("Panjang Queue : " + queue.size());
            queue.showQueue();
            System.out.println();
```

➔ **Melihat ukuran queue, dan menampilkan seluruh nilai queue**

```
            System.out.println("Nilai Dequeue :" + queue.deQueue());
            System.out.println("Panjang Queue :" + queue.size());
            System.out.println("-------------------");
            queue.showQueue();
            System.out.println();
```

➔ **Mencetak hasil pengambilan nilai queue, dan ukuran queue setelahnya, kemudian menampilkan seluruh isi queue**

```
        }
}
```

BAB III
PENUTUP

2.1 Kesimpulan

Abstract Data Type (ADT) adalah kumpulan dari elemen-elemen data yang disajikan dengan satu set operasi yang digambarkan pada elemen-elemen data tersebut. Stacks, queues dan binary trees adalah tiga contoh dari ADT.

Stack adalah satu set atau urutan elemen data dimana manipulasi data dari elemen-elemen hanya diperbolehkan pada tumpukan teratas dari stack. Hal ini merupakan perintah pengumpulan data secara linier yang disebut “Last In, First Out” (LIFO).

Queues adalah contoh lain dari ADT. Hal ini merupakan perintah pengumpulan data yang disebut “First In, First Out”. Enqueue dan dequeue merupakan operasi yang dihubungkan dengan queues. Enqueu menunjuk pada memasukkan data pada akhir queue dimana dequeue berarti memindahkan elemen dari queue tersebut.

Linked list merupakan struktur data dinamis yang berlawanan dengan array, yang merupakan struktur statis. Hal ini berarti linked list dapat tumbuh dan berkurang dalam size yang bergantung pada kebutuhan user.

ADT biasanya dapat diwakilkan menggunakan sequential dan linked representation. Hal ini memudahkan untuk membuat sequential representation dengan menggunakan array. Bagaimanapun juga, masalah dengan menggunakan array adalah pembatasan size, yang membuatnya tidak fleksibel. Dengan menggunakan array, sering terjadi kekurangan atau kelebihan space memory. Mempertimbangkan hal tersebut, Anda harus membuat sebuah array dan mendeklarasikannya agar mampu menyimpan 50 elemen. Jika user hanya memasukkan 5 elemen, maka 45 space pada memory akan sia-sia. Disisi lain, jika user ingin memasukkan 51 elemen, space yang telah disediakan didalam array tidak akan cukup.

Dibandingkan dengan sequential representation, linked representation lebih sedikit rumit tetapi lebih fleksibel. Linked representation menyesuaikan memory yang dibutuhkan oleh user.

DAFTAR PUSTAKA

[1] Thamura, Frans. *Modul JENI-intro3: Bab03 Teknik Pemrograman Lanjut*. JENI.

[2] Gustian Ri'pi. Contoh Program ADT pada Java. https://blog.ub.ac.id/gustiangr8/2014/03/contoh-program-adt-pada-java/. Universitas Brawijaya. 06 Maret 2014.

[3] Maulana Adieb. Memahami Stack. https://glints.com/id/lowongan/stack-adalah/. 04 Februari 2021.

[4] Agung Setiawan. Implementasi Struktur Data Stack. https://agung-setiawan.com/implementasi-struktur-data-stack-java/. 19 Januari 2014.

[5] Jejaring. Contoh Program Java Queue. https://www.jejaring.web.id/contoh-program-java-queue/. 26 Februari 2019.

[6] Bobby Syahronanda. Contoh Program Queue dengan Java. http://bobby-syahronanda.blogspot.com/2014/01/contoh-program-queue-dengan-java.html. 18 Januari 2014.

[7] Semut Belang. Definisi dan Contoh Queue. https://semutaspal.com/queue/. 29 November 2019.

[8] Bundet. Pengertian Queue. https://bundet.com/d/697-pengertian-queue-antrian. 23 September 2020.

[9] Fidelson Tanzil S.KOM., M.T.I. Queue. https://socs.binus.ac.id/2018/12/21/queue/. Binus University. 21 Desember 2018.

[10] Zalmi Randa Ramadhan. Pengertian Linked List. https://www.researchgate.net/publication/332717356_Pengertian_linked_list/. April 2019.